Theoretical Computer Science 72 (1990) 203–223
North-Holland

A²CCS: ATOMIC ACTIONS FOR CCS*

Roberto GORRIERI
Dipartimento di Informatica, Università di Pisa, Corso Italia 40, I-56100 Pisa, Italy

Sergio MARCHETTI**
Selenia S.p.A., Via Tiburtina km. 14.7, I-00100 Roma, Italy

Ugo MONTANARI
Dipartimento di Informatica, Università di Pisa, Corso Italia 40, I-56100 Pisa, Italy

Abstract. An operational model of atomic actions based on compositional transition systems is presented. A notion of abstraction between lower level and higher level transition systems is defined, where actions in the higher level correspond to sequences of actions in the lower level. Abstraction and composition are proved to commute. The approach is applied to an extended version of CCS, called A²CCS, where a prefix operation containing an underlined action symbol expresses the fact that the action, when executed, must be part of a sequence of actions performed in an atomic manner.

1. Introduction

Milner's Calculus for Communicating Systems is a basic specification language for describing communicating processes, with a simple and well-defined operational semantics [12]. It has been a touchstone for many of the proposed theories of concurrency.

A specification language of practical relevance is usually required to possess suitable abstraction and modularization mechanisms, or at least to allow their straightforward definition. While in a sense this is not a problem for CCS agents, which are intrinsically compositional, the actions agents perform cannot be abstracted, i.e. they cannot be easily considered at different levels of detail. See for instance [7], where this and other open problems in concurrency are mentioned.

As an example, let us consider the following remark, presented in [6] in a different context. Given the CCS agent $p = \alpha\mathrm{NIL}|\beta\mathrm{NIL}$, let us assume that actions α and β are decomposed at a lower level of observation into sequences $\alpha'\alpha''$ and $\beta'\beta''$, respectively. We thus obtain the agent $p' = \alpha'\alpha''\mathrm{NIL}|\beta'\beta''\mathrm{NIL}$. One would expect

* Partially supported by Selenia S.p.A.
** Research performed at Dipartimento di Informatica, Università di Pisa.

that the behaviour of p, where α and β have been replaced with $\alpha'\alpha''$ and $\beta'\beta''$, be equal to, or at least be equivalent to, the behaviour of p'. But this is not the case, even for trace equivalence: the trace set of p is $\{\alpha\beta, \beta\alpha\}$, which after the substitution becomes $\{\alpha'\alpha''\beta'\beta'', \beta'\beta''\alpha'\alpha''\}$, while the trace set of p' is

$$\{\alpha'\alpha''\beta'\beta'', \beta'\beta''\alpha'\alpha'', \alpha'\beta'\alpha''\beta'', \alpha'\beta'\beta''\alpha'', \beta'\alpha'\beta''\alpha'', \beta'\alpha'\alpha''\beta''\}.$$

The apparent paradox is due to the fact that CCS actions are considered as atomic, namely their execution cannot be interleaved with the execution of concurrent actions. Still, the fact remains that there is no easy way in CCS to express the requirement that sequences $\alpha'\alpha''$ and $\beta'\beta''$ in p' be executed as atomic actions.

The fact that certain sequences of actions must be executed atomically is not a completely alien concept in CCS. The operators of nondeterministic choice and recursion have a similar implicit requirement. The inference rule (Sum) of CCS

$$\text{(Sum)} \quad E' \xrightarrow{\mu} E'' \text{ implies } E + E' \xrightarrow{\mu} E'' \text{ and } E' + E \xrightarrow{\mu} E''$$

implies that (i) choice between left and right alternatives and (ii) execution of an action, which are conceptually two distinct steps, are made strictly in the same temporal moment, i.e. in an atomic manner. Similarly, rule

$$\text{(Rec)} \quad E'[\text{rec } x.E'/x] \xrightarrow{\mu} E'' \text{ implies } \text{rec } x.E' \xrightarrow{\mu} E''$$

implies that a sequence of a recursion unfolding and of an action execution is atomic. Note also that sequences of any length of choices and of recursion unfoldings together, terminating with an action, are atomic. To make the point more clear, let us consider a version of CCS where rules (Sum) and (Rec) are replaced by axioms like

$$E + E' \overset{\pi}{\rightarrowtail} E'$$

$$E' + E \overset{\pi}{\rightarrowtail} E'$$

$$\text{rec } x.E \overset{\pi}{\rightarrowtail} E[\text{rec } x.E/x]$$

where π is an invisible action, and where somehow an agent which has executed a π action cannot be interrupted by concurrent agents. It is possible to see that the new and the classical versions of CCS coincide, provided that π moves are disregarded in the behaviours of agents.

The details of the construction are as follows. Whenever an agent performs an action π, it enters a special marked state. A marked agent has priority, i.e. it cannot stay idle when composed with an agent performing an action different from π. A marked agent is invisible, in the sense that a whole computation from an unmarked state to an unmarked state through marked states can be shown to correspond to a single transition in Milner's system. The rules are:

(Act) $\mu E \overset{\mu}{\rightarrowtail} E$,

(Res) $E' \overset{\gamma}{\rightarrowtail} E''$ and $\gamma \neq \alpha, \alpha^-$ implies $E'\backslash\alpha \overset{\gamma}{\rightarrowtail} E''\backslash\alpha$,

(Rel) $E' \overset{\gamma}{\rightarrowtail} E''$ implies $E'[\phi] \overset{\phi(\gamma)}{\longrightarrowtail} E''[\phi]$,

(Sum') $E + E' \overset{\pi}{\rightarrowtail} {}^*E'$, $\quad E' + E \overset{\pi}{\rightarrowtail} {}^*E'$,

(Com') $E' \overset{\gamma}{\rightarrowtail} E''$ and $\gamma = \pi$ or E not marked

implies $E'|E \overset{\gamma}{\rightarrowtail} E''|E$ and $E|E' \overset{\gamma}{\rightarrowtail} E|E''$,

$E_1' \overset{\lambda}{\rightarrowtail} E_1''$ and $E_2' \overset{\lambda^-}{\longrightarrow} E_2''$

implies $E_1'|E_2' \overset{\tau}{\rightarrowtail} E_1''|E_2''$,

(Rec') $\mathrm{rec}\, x.E \overset{\pi}{\rightarrowtail} {}^*E[\mathrm{rec}\, x.E/x]$,

(Close) $E' \overset{\gamma}{\rightarrowtail} E''$ implies ${}^*E' \overset{\gamma}{\rightarrowtail} E''$

where $\phi(\pi) = \pi$ and γ stays for μ or π.

For instance, the agent $(\alpha\mathrm{NIL} + \mathrm{NIL})|(\alpha^-\mathrm{NIL} + \mathrm{NIL})$ may exhibit the following sequence of moves:

$$(\alpha\mathrm{NIL} + \mathrm{NIL})|(\alpha^-\mathrm{NIL} + \mathrm{NIL}) \overset{\pi}{\rightarrowtail} {}^*\alpha\mathrm{NIL}|(\alpha^-\mathrm{NIL} + \mathrm{NIL})$$

$$\overset{\pi}{\rightarrowtail} {}^*\alpha\mathrm{NIL}|{}^*\alpha^-\mathrm{NIL} \overset{\tau}{\rightarrowtail} \mathrm{NIL}|\mathrm{NIL}$$

corresponding to Milner's transition

$$(\alpha\mathrm{NIL} + \mathrm{NIL})|(\alpha^-\mathrm{NIL} + \mathrm{NIL}) \overset{\tau}{\longrightarrow} \mathrm{NIL}|\mathrm{NIL}.$$

However the sequence

$$(\alpha\mathrm{NIL} + \mathrm{NIL})|(\alpha^-\mathrm{NIL} + \mathrm{NIL}) \overset{\pi}{\rightarrowtail} {}^*\alpha\mathrm{NIL}|(\alpha^-\mathrm{NIL} + \mathrm{NIL})$$

$$\overset{\pi}{\rightarrowtail} {}^*\alpha\mathrm{NIL}|{}^*\alpha^-\mathrm{NIL} \overset{\alpha}{\rightarrowtail} \mathrm{NIL}|{}^*\alpha^-\mathrm{NIL}$$

is not possible, since in the last move, rule (Com') would not apply. In fact, the above sequence would correspond to the deduction

$$(\alpha\mathrm{NIL} + \mathrm{NIL})|(\alpha^-\mathrm{NIL} + \mathrm{NIL}) \overset{\alpha}{\longrightarrow} \mathrm{NIL}|\alpha^-\mathrm{NIL}$$

which is not possible using Milner rules. Notice that there are sequences of π moves which terminate in a deadlocked state: they do not correspond to any move in

Milner's system (they may be said to correspond to *failed proofs*). For instance

$$(\alpha\mathrm{NIL}+\mathrm{NIL})|(\alpha\mathrm{NIL}+\mathrm{NIL}) \overset{\pi}{\rightarrowtail} {}^*\alpha\mathrm{NIL}|(\alpha\mathrm{NIL}+\mathrm{NIL})$$

$$\overset{\pi}{\rightarrowtail} {}^*\alpha\mathrm{NIL}|{}^*\alpha\mathrm{NIL}.$$

A number of considerations are in order.

(1) Sequences of actions performed in an atomic manner are hidden in CCS semantics. In particular, the existence of infinite sequences accounts for the unbounded nondeterminism of unguarded CCS (for instance

$$\mathrm{rec}\ x.\alpha\mathrm{NIL}|x \xrightarrow{\alpha} \alpha\mathrm{NIL}|\ldots|\alpha\mathrm{NIL}|\mathrm{NIL}|\mathrm{rec}\ x.\alpha\mathrm{NIL}|x$$

for any number of αNIL agents in the composition).

(2) The ability of specifying atomic sequences of actions is a very powerful mechanism in a language, and may extend its semantics in unexpected ways. For instance, the type of nondeterminism expressed by (Sum') above is very elementary: just local choice. However, the atomicity condition transforms it into the sophisticated, symmetric, global nondeterminism of standard CCS. Similar considerations are suggested by [4, 5]: a rather simple schema where processes access in turn a hierarchy of shared data is capable of expressing full synchronization when equipped with an atomic action mechanism.

(3) The syntax-directed style of CCS definition works very well in handling atomicity. In fact, both Milner rules and those given above are simple and easy to understand. However, the result achieved in the construction above is not insignificant: it amounts more or less to a hierarchical implementation of the CCS communication mechanism in terms of a much lower level mechanism based on independent communication choices. Moreover, the nondeterminism of the logical metalanguage describing the dynamics of CCS accounts for the painless handling of failed transitions in Milner's system, since failed transitions simply correspond to failed, goal oriented proofs. Instead, in the more detailed implementation system, a failed transition can also show up as a sequence of moves leading to a particular marked deadlocked state. Of course, such sequences of moves can be ruled out by considering only computations from, and to, unmarked states. Alternatively, it would be possible to define a more "intelligent" interpreter of the language at the implementation level, which can backtrack from such deadlocked states, thus obtaining a more deterministic implementation.

The above construction and considerations were the starting point of the research described in this paper. However, we no longer elaborate on such an implementation calculus for CCS. Rather, we have set for ourselves the goal of extending the hidden ability of CCS of handling atomic sequences by providing explicit constructs. By analogy with (Sum') above, we wanted a construct expressing the fact that an atomic sequence was starting with the present action. The syntactic extension needed to accommodate this new construct turned out to be minimal: the prefix operation μE

may be assembled also with an underlined action: $\underline{\mu}E$. The meaning of $E' = \underline{\mu}E$ is that E' can execute action μ only as the beginning (or, in general, only as part) of an atomic sequence. For instance, the agent p' above would become in our syntax:

$$p'' = \underline{\alpha}'\alpha''\mathrm{NIL} | \underline{\beta}'\beta''\mathrm{NIL}$$

with the understood semantics that only traces $\alpha'\alpha''\beta'\beta''$ and $\beta'\beta''\alpha'\alpha''$ should be possible.

Of our extended version of CCS (which we call Atomic Action CCS, or A^2CCS) we give two semantics, in the usual form of transition systems: a lower level semantics, where transitions are labelled by regular CCS actions, but where certain states are marked, i.e. invisible; and an upper level semantics, where states are regular A^2CCS terms but where transitions are labelled with strings of CCS actions. These strings could be considered as denoting more abstract actions. Of course both transition systems coincide with the classical CCS transition system whenever no underlined prefixes are present.

For instance, in our example we have the following sequence in the lower transition system

$$\underline{\alpha}'\alpha''\mathrm{NIL} | \underline{\beta}'\beta''\mathrm{NIL} \overset{\alpha'}{\rightsquigarrow} *\alpha''\mathrm{NIL} | \underline{\beta}'\beta''\mathrm{NIL} \overset{\alpha''}{\rightsquigarrow} \mathrm{NIL} | \underline{\beta}'\beta''\mathrm{NIL}$$

$$\overset{\beta'}{\rightsquigarrow} \mathrm{NIL} | *\beta''\mathrm{NIL} \overset{\beta''}{\rightsquigarrow} \mathrm{NIL} | \mathrm{NIL}$$

and we have the following sequence in the higher transition system

$$\underline{\alpha}'\alpha''\mathrm{NIL} | \underline{\beta}'\beta''\mathrm{NIL} \xrightarrow{\alpha'\alpha''} \mathrm{NIL} | \underline{\beta}'\beta''\mathrm{NIL} \xrightarrow{\beta'\beta''} \mathrm{NIL} | \mathrm{NIL}.$$

The correspondence between lower and higher semantics is straightforward. Visible, i.e. unmarked states in the lower level are in (the identity) bijection with states in the upper level, while a sequence of transitions from an unmarked state via marked states to an unmarked state in the lower level (we call it a *metatransition*) corresponds to a transition in the higher level. In our example, the metatransition

$$\underline{\alpha}'\alpha''\mathrm{NIL} | \underline{\beta}'\beta''\mathrm{NIL} \overset{\alpha'}{\rightsquigarrow} *\alpha''\mathrm{NIL} | \underline{\beta}'\beta''\mathrm{NIL} \overset{\alpha''}{\rightsquigarrow} \mathrm{NIL} | \underline{\beta}\ \beta''\mathrm{NIL}$$

corresponds to the transition

$$\underline{\alpha}'\alpha''\mathrm{NIL} | \underline{\beta}'\beta''\mathrm{NIL} \xrightarrow{\alpha'\alpha''} \mathrm{NIL} | \underline{\beta}'\beta''\mathrm{NIL}.$$

The correspondence between transition systems defined in this way is called an *abstraction*. By considering abstractions as independent from CCS, we discovered that our construction based on marked processes and their priority was rather general: it only assumed a CCS-like notion of composition. Thus in the paper we start our presentation in this general setting, and we prove the fundamental property that composition and abstraction commute: if we start with transition systems T'_0

and T''_{lo} and consider abstractions T'_h of T'_{lo} and T''_h of T''_{lo}, the composition $T'_h | T''_h$ is an abstraction of composition $T'_{lo} | T''_{lo}$.

A^2CCS is a rather powerful language. The classical dining philosophers example at the end of the paper shows that A^2CCS has acquired for instance the ability of handling multiple synchronizations (a triple synchronization, in the case of a philosopher and two forks), thus avoiding deadlocks.

The choice of the underlined prefix for expressing atomic sequencing was clearly suggested by our line of thought. A possible alternative could be a rule like

$$p \xrightarrow{u} p' \equiv 1 \text{ and } q \xrightarrow{v} q' \quad \text{implies} \quad p; q \xrightarrow{u;v} q'$$

proposed in [5]. While we did not consider the relative power of the two constructs, we find that our proposal is more in the traditional, tail-recursive CCS style.

2. High level and low level transition systems

We want to describe the same system at two different levels of abstraction.

(1) At *high level*, atomic behaviours are simply described by transitions, and thus only the observable features are considered: initial, final states and the *sequence* of actions performed. At high level we relax the assumption that an action associated to a transition has to be considered *elementary*, i.e. not decomposable in finer actions.

(2) At *low level*, implementative details are explicitly taken into account. Atomic behaviours are modelled as sequences of adjacent transitions, called metatransitions, and therefore we need suitable transition rules to guarantee the atomicity property.

Now we give a list of six requirements we impose on the definition of high and low level transition systems and on the notion of abstraction and composition.

High level transition systems

(i) Intermediate states are not considered: thus transitions are automatically atomic. To every transition we associate a nonempty sequence of elementary actions, the last of which is called *commit*.

Low level transition system

(ii) There exists a set of *visible* states. The remaining states are called *invisible*. The initial state must be visible.

(iii) Metatransitions are described as sequences of adjacent transitions, each labelled with an elementary action, with the assumption that the intermediate states are invisible and the extreme states are visible. Note that a transition from a visible state to a visible state is considered a metatransition as well.

Abstraction

A high level transition system is an *abstraction* of a low level transition system (or equivalently a low level transition system is an *implementation* of a high level transition system) if the following conditions hold.

(iv) There exists a bijective mapping between the visible states of the lower transition system and the states of the higher transition system; this mapping must preserve the initial state.

(v) There exists a surjective mapping from lower metatransitions to higher transitions. This mapping must preserve the associated sequences of actions and the extreme states (when mapped via the bijection in (iv)).

Let us define formally the notion of abstraction.

Definition 2.1. A *transition system* S, labelled on an alphabet K, is a triple, $S = \langle Q, q, T\rangle$, where

- Q is a set of *states*;
- q is the *initial* state;
- $T \subseteq Q \times K \times Q$ is a set of *transitions*.

Definition 2.2. Given two transition systems $S_{lo} = \langle Q_{lo}, q_{lo}, T_{lo}\rangle$, labelled on A, and $S_h = \langle Q_h, q_h, T_h\rangle$, labelled on A^+, we say that S_h is an *abstraction* of S_{lo} iff there exists a homomorphism $h = \langle h_Q, h_T\rangle$ from S_{lo} to S_h, $h: S_{lo} \Rightarrow S_h$, such that

- $h_Q: Q_{lo} \to Q_h$ is a partial injective and surjective function, where $\mathbf{Dom}(h_Q) = h_Q^{-1}(Q_h) = V_{lo}$ is the set of lower *visible* states;
- a *metatransition* m is a sequence of transitions, $m = t_1 t_2 \ldots t_n$, where $t_i \in T_{lo}$ and $t_i = \langle q_i', \mu_i, q_i''\rangle$, such that $q_i'' = q_{i+1}'$, $i = 1, \ldots, n-1$ (transitions are adjacent), q_1', $q_n'' \in V_{lo}$ (initial and final states are visible), and $q_i' \in Q_{lo} \setminus V_{lo}$, $i = 2, \ldots, n$ (intermediate states are invisible). M_{lo} is the set of all metatransitions in S_{lo};
- $h_T: M_{lo} \to T_h$ is a surjective function such that $h_T(\langle q_1', \mu_1, q_1''\rangle\langle q_2', \mu_2, q_2''\rangle \ldots \langle q_n', \mu_n, q_n''\rangle) = \langle q', s, q''\rangle$, where $q' = h_Q(q_1')$, $q'' = h_Q(q_n'')$ and $s = \mu_1\mu_2 \ldots \mu_n$.

3. Composition of transition systems

Now we consider structured systems, i.e. composed of subsystems. Thus an operation "|" of parallel composition must be defined on transition systems. We require that composition be defined in such a way that the following commutativity property holds.

(vi) Abstraction and composition commute, i.e. to compose two low level transition systems and then to abstract the result must be the same as to abstract the two low level transition systems and then to compose them. Formally, given homomorphisms $h': S_{lo}' \Rightarrow S_h'$ and $h'': S_{lo}'' \Rightarrow S_h''$, then there exists a homomorphism $h: S_{lo} \Rightarrow S_h$, where $S_{lo} = S_{lo}' |_{lo} S_{lo}''$ and $S_h = S_h' |_h S_h''$.

Here we restrict ourselves to a particular class of transition systems and of composition operators. At high level, the set of states of the composed system is simply the Cartesian product of the sets of states of the components. The initial state is the pair of the initial states, and the transitions are defined by the following

inference rules:

$$\begin{aligned} &q_1 \xrightarrow{s} q_2 \quad \text{implies} \quad [q_1, q] \xrightarrow{s} [q_2, q] \text{ and } [q, q_1] \xrightarrow{s} [q, q_2], \\ &q_1' \xrightarrow{s'} q_2' \text{ and } q_1'' \xrightarrow{s''} q_2'' \text{ and } S\text{-syn}(s', s'', s) \\ &\quad \text{implies} \quad [q_1', q_1''] \xrightarrow{s} [q_2', q_2''] . \end{aligned} \tag{3.1}$$

Here, variable s denotes a sequence of elementary actions, and relation S-syn(s', s'', s) represents a condition for the synchronization of s' and s'' to form s, i.e. for both components to cooperate within the same transition. Relation S-syn is defined as follows:

$$\begin{aligned} &\text{syn}(\mu', \mu'', \mu) \quad \text{implies} \quad S\text{-syn}(\mu', \mu''s, \mu s) \text{ and } S\text{-syn}(\mu's, \mu'', \mu s), \\ &S\text{-syn}(s', s'', s) \quad \text{implies} \quad S\text{-syn}(\mu s', s'', \mu s) \text{ and } S\text{-syn}(s', \mu s'', \mu s), \\ &S\text{-syn}(s', s'', s) \text{ and } \text{syn}(\mu', \mu'', \mu) \quad \text{implies} \quad S\text{-syn}(\mu's', \mu''s'', \mu s). \end{aligned} \tag{3.2}$$

Here syn(μ', μ'', μ) defines the synchronization relation of the elementary actions: μ' synchronizes with μ'' giving μ as the result. Note the requirement that the commit of the subtransition which ends first has to be synchronized with an action of the other subtransition. From this definition it is clear that the commit of a transition is unique and is "caused" by all the actions in the transition.

At low level, we assume that to express the fact that a state is invisible we add a mark * to it. Moreover, we let the visible states coincide with the states of the high level system, thus guaranteeing (iv) above. Given two components in states q' and q'', the state of the composed system is either $[q', q'']$ or $[*q', q'']$ or $[q', *q'']$. A state q of a composed system is visible iff $q = [q', q'']$ and both q' and q'' are visible. Thus the set of visible states of the composed system is the Cartesian product of the visible states of the components.

To define the transitions of the composed systems, we need two auxiliary predicates P (for asynchrony) and Q (for synchrony) which depend only on their arguments being invisible (i.e. marked) states or not. They are defined as shown in Fig. 1(a) and (b), respectively. The transitions are as follows.

$$\begin{aligned} &q_1 \xrightarrow{\mu} q_2 \text{ and } P(q_1, q_2, q, x) \\ &\quad \text{implies} \quad [q_1, q] \xrightarrow{\mu} [q_2, xq] \text{ and } [q, q_1] \xrightarrow{\mu} [xq, q_2], \\ &q_1' \xrightarrow{\mu'} q_2' \text{ and } q_1'' \xrightarrow{\mu''} q_2'' \text{ and } \text{syn}(\mu', \mu'', \mu) \text{ and } Q(q_1', q_1'') \\ &\quad \text{implies} \quad [q_1', q_1''] \xrightarrow{\mu} [q_2', q_2''] \end{aligned} \tag{3.3}$$

where x may be $*$ or empty. Note that predicate P can turn the state of the idle subsystem to invisible when $x = *$.

From Fig. 1(a) we can see that if the idle subsystem is visible, the other subsystem can always move asynchronously (entries $-/\mathbb{T}$, $x =$ empty). Furthermore, if the

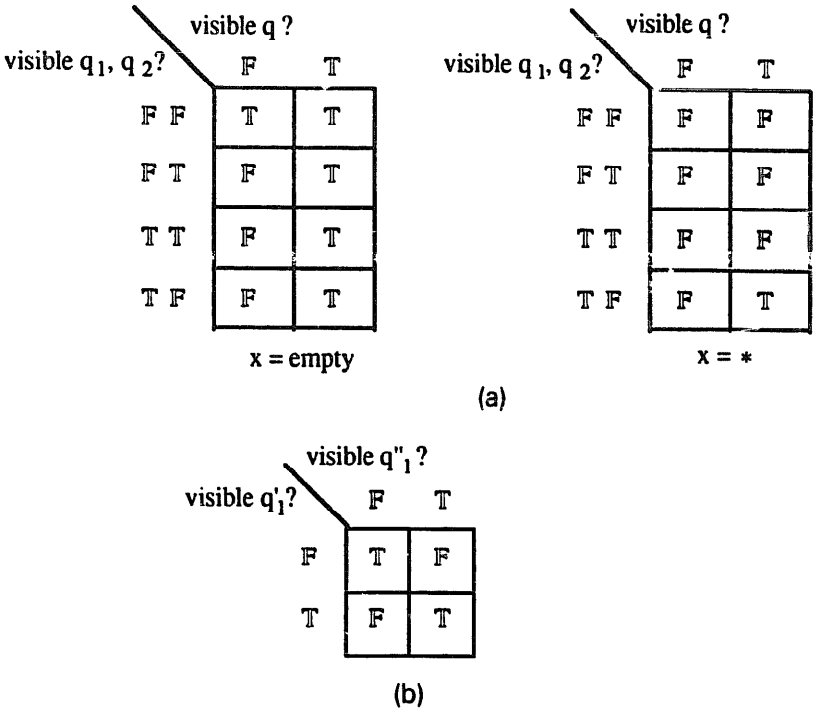

Fig. 1. The definitions of predicates (a) $P(q_1, q_2, q, x)$ and (b) $Q(q_1, q_1')$ of (3.3).

moving subsystem is starting its metatransition (entries $\mathbb{TF}/\mathbb{T}$, $x = -$), it may, or may not, turn the state of the idle subsystem to invisible. If the idle subsystem is invisible, the other can move only if it executes an intermediate step of a metatransition (entry $\mathbb{FF}/\mathbb{F}$, $x =$ empty). Thus, if a system is invisible, its visible subsystems (if any) are blocked until the system state becomes visible.

From Fig. 1(b), synchronizations between two transitions are possible iff both transitions are starting either from two visible or from two invisible states (entries $\mathbb{F}/\mathbb{F}$ and $\mathbb{T}/\mathbb{T}$). Note that the synchronization of two transitions, both going from a visible state to an invisible state, is another way to start the concurrent execution of two metatransitions.

With the above assumptions we are able to prove the following theorem, which makes sure that requirement (vi) is satisfied. A lemma is necessary.

Lemma 3.1. *Let S'_{lo}, S''_{lo}, S'_h, S''_h be four transition systems, and let $h'\colon S'_{lo} \Rightarrow S'_h$ and $h''\colon S''_{lo} \Rightarrow S''_h$ be abstraction homomorphisms. Let $S_{lo} = S'_{lo} \|_{lo} S''_{lo}$. Given a metatransition $m \in M_{lo}$, $m = [q'_0, q''_0] \to^{\mu_1} [q'_1, q''_1] \ldots [q'_{n-1}, q''_{n-1}] \to^{\mu_n} [q'_n, q''_n]$, the following properties hold:*

(i) *if there exists an index i, $0 < i < n$, such that q'_i is visible, then this component stays idle in the successive transitions of the metatransition;*

(ii) *if $[q'_0, q''_0] \to^{\mu_1} [q'_1, q''_1]$ is proved by the second (3.3) rule or by the first (3.3) rules with $x = *$, then m can be considered the composition of two metatransitions m' and m''; conversely, if transition $[q'_0, q''_0] \to^{\mu_1} [q'_1, q''_1]$ is proved by the first (3.3) rule with $x =$ empty, then one of the two components stays idle in every step.*

(iii) *if m is obtained by the composition of two metatransitions, then there exists one and only one transition $[q'_i, q''_i] \rightarrow^{\mu_{i+1}} [q'_{i+1}, q''_{i+1}]$, such that either q'_{i+1} is visible, the proof is by the second (3.3) rule, and the first agent stays idle in the successive steps (if any); or the symmetric property holds (or both).*

Proof. (i) If a subagent is invisible and the other is visible, then the visible subagent can never move, neither asynchronously (entries $\mathbb{T}-/\mathbb{F}, x=-$) nor synchronously (entries $\mathbb{T}/\mathbb{F}$ and $\mathbb{F}/\mathbb{T}$). Besides, its state cannot be turned to invisible, since entries $\mathbb{F}-/\mathbb{T}$, $x=*$ are false.

(ii) If $[q'_0, q''_0] \rightarrow^{\mu_1} [q'_1, q''_1]$ is proved by the second (3.3) rule, then both metatransitions take part in the first step. If it is proved by the first (3.3) rule with $x=*$, then both q'_1 and q''_1 are invisible; since an agent cannot become visible without moving, both components will move. If the first (3.3) rule with $x=$ empty is used, then one of the two components stays idle during the first transition, and after it remains visible. By (i) above, it will still stay idle in every step.

(iii) From Fig. 1(a), we can see that a subagent can never end its metatransition asynchronously if the other is invisible (entries $\mathbb{FT}/\mathbb{F}$, $x=-$). Thus when the shortest metatransition terminates, the transition with the above property takes place. □

Theorem 3.2 (Abstraction and composition commute). *Given transition systems S'_{lo}, S''_{lo}, S'_h, S''_h and abstraction homomorphisms h' and h'', $h': S'_{lo} \Rightarrow S'_h$, $h'': S''_{lo} \Rightarrow S''_h$, let $S_{lo} = S'_{lo}|_{lo} S''_{lo}$ and $S_h = S'_h|_h S''_h$. In this hypothesis, there exists a homomorphism $h: S_{lo} \Rightarrow S_h$.*

Proof. Bijective function $h_Q: V'_{lo} \times V''_{lo} \rightarrow Q'_h \times Q''_h = h'_Q(V'_{lo}) \times h''_Q(V''_{lo})$ is defined as $h_Q([q', q'']) = [h'_Q(q'), h''_Q(q'')]$. This is consistent with our construction for $|$. Note that h_Q preserves the initial state, since it is visible. The existence of surjective function $h_T: M_{lo} \rightarrow T_h$ is given constructively.

Let m be a metatransition, $m = q_0 \rightarrow^{\mu_1} q_1 \ldots q_{n-1} \rightarrow^{\mu_n} q_n$, where $q_i = [q'_i, q''_i]$. The corresponding transition t in $S'_h|_h S''_h$ that we want to construct has $h_Q([q'_0, q''_0])$ as initial state and $h_Q([q'_n, q''_n])$ as final state. There are two cases:

(i) m is due to the asynchronous moving of one of the two subagents: Let us suppose, for instance, that the left component is active. Therefore, $q''_0 = q''_1 = \cdots = q''_n$. Furthermore, $q_0 \rightarrow^{\mu_1} q_1 \ldots q_{n-1} \rightarrow^{\mu_n} q_n$ is generated by repeated applications of the first (3.3) rule (first consequent). Thus, we can derive metatransition $m' = q'_0 \rightarrow^{\mu_1} q'_1 \ldots q'_{n-1} \rightarrow^{\mu_n} q'_n$, and by h'_T its corresponding transition is $t' = h'_Q(q'_0) \rightarrow^{\mu_1 \mu_2 \cdots \mu_n} h'_Q(q'_n)$. By the first (3.1) rule (first consequent), transition t is derived: $t = [h'_Q(q'_0), h''_Q(q''_0)] \rightarrow^{\mu_1 \mu_2 \cdots \mu_n} [h'_Q(q'_n), h''_Q(q''_n)]$.

(ii) m is a synchronization of two metatransitions: The construction of $h_T(m)$ is by induction on the length of m, and proceeds by scanning the sequence of its transitions. The rule involved in the proof of the corresponding transition in S_h is the second (3.1) rule, but the core of the proof is in establishing a correspondence

between (proofs of) lower transitions and proof steps for S-syn. As the basic step of the induction, let us consider the transition $[q'_i, q''_i] \to^{\mu_{i+1}} [q'_{i+1}, q''_{i+1}]$, proved by the second (3.3) rule and where, for instance, q'_{i+1} is visible. This transition does exist and is unique by Lemma 3.1(iii). The sequence of transitions $q_i \to^{\mu_{i+1}} q_{i+1} \ldots q_{n-1} \to^{\mu_n} q_n$, corresponds to the first (3.2) rule, with $s = \mu_{i+2} \ldots \mu_n$ and $\mu = \mu_{i+1}$. For the inductive step, if a lower transition is proved by the first (3.3) rule, then the corresponding proof step for S-syn is obtained by the second (3.2) rule; if the transition is proved by the second (3.3) rule, then S-syn is proved by the third (3.2) rule.

To prove that h_T is surjective, given a transition $t = h_Q(q_0) \to^{\mu_1 \mu_2 \ldots \mu_n} h_Q(q_n)$ in S_h, we exhibit a corresponding metatransition m by reversing the construction above. □

4. A²CCS

In the following, we apply our approach to a well-known language, Milner's Calculus for Communicating Systems (CCS), whose operational semantics is based on a transition system [12]. We introduce a syntactic and semantic extension of CCS which we call Atomic Action CCS (A²CCS) and for the latter language we define a high level and a low level transition system, which satisfy requirements (i)-(vi) above.

Definition 4.1 (*Syntactical extension of CCS*)

- Δ, ranged over by α, is the set of *basic* actions and $\Delta^- = \{\alpha^- \mid \alpha \in \Delta\}$;
- $\Lambda = \Delta \cup \Delta^-$, ranged over by λ, is the set of *observable* actions;
- $M = \Lambda \cup \{\tau\}$, ranged over by μ, is the set of *elementary* actions;
- $\underline{M} = \{\underline{\mu} \mid \mu \in M\}$ is the set of *private prefixes*;
- $\Sigma = M \cup \underline{M}$, ranged over by σ, is the set of *prefixes*.

The *A²CCS agents* (or *states*) are closed terms (i.e. terms without free variables) which can be generated by the following BNF-like grammar

$$F ::= x \mid \mathrm{Nil} \mid \sigma F \mid F \backslash \alpha \mid F[\phi] \mid F + F \mid F \mid F \mid \mathrm{rec}\, x.F,$$

where x is a variable and ϕ is a permutation of M which preserves τ and the operation $^-$ of complementation.

The *CCS agents* have the same syntax, except for the prefix operator that is defined only for the elementary actions.

Definition 4.2 (*Synchronization relation for the elementary actions*). Relation syn for CCS is defined as follows: for every λ, $\mathrm{syn}(\lambda, \lambda^-, \tau)$ is true.

The synchronization relation for sequences of elementary actions, ranged over by s, is defined in (3.2).

Definition 4.3 (*High level transition system for A²CCS*). Our *high level A²CCS derivation relation* $F_1 \rightarrow^s F_2$ over A²CCS agents is defined as the least relation satisfying the following axiom and inference rules:

(act) $\mu F \xrightarrow{\mu} F$,

(seq) $F_1 \xrightarrow{s} F_2$ implies $\mu F_1 \xrightarrow{\mu s} F_2$,

(res) $F_1 \xrightarrow{s} F_2$ and α, α^- do not appear in s implies $F_1\backslash\alpha \xrightarrow{s} F_2\backslash\alpha$,

(rel) $F_1 \xrightarrow{s} F_2$ implies $F_1[\phi] \xrightarrow{\phi(s)} F_2[\phi]$,

(sum) $F_1 \xrightarrow{s} F_2$ implies $F + F_1 \xrightarrow{s} F_2$ and $F_1 + F \xrightarrow{s} F_2$,

(com) $F_1 \xrightarrow{s} F_2$ implies $F|F_1 \xrightarrow{s} F|F_2$ and $F_1|F \xrightarrow{s} F_2|F$,

$F_1 \xrightarrow{s} F_2$ and $F_1' \xrightarrow{s'} F_2'$ and S-syn(s, s', s'')

implies $F_1|F_1' \xrightarrow{s''} F_2|F_2'$.

(rec) $F_1[\mathrm{rec}\, x.F_1/x] \xrightarrow{s} F_2$ implies $\mathrm{rec}\, x.F_1 \xrightarrow{s} F_2$.

Note that ϕ is obviously extended to sequences and that (com) rules coincide with (3.1). Note also that rules (act), (res), (rel), (sum), (com) and (rec), letting $s = \mu$ and thus S-syn = syn, coincide with CCS inference rules. We define now the low level transition system for A²CCS.

Definition 4.4 (*Low level visible and invisible states*). The A²CCS *low level agents* L are the closed terms which can be generated by the following BNF-like grammar

$$L ::= F \mid *L \mid L\backslash\alpha \mid L[\phi] \mid L|L,$$

where F is an A²CCS agent. From the syntax we see that the agents F (i.e. A²CCS closed terms) are also low level agents. They are the *visible* low level agents. A²CCS invisible low level agents are ranged over by I.

Definition 4.5 (*Low level transition system for A²CCS*). Our *low level derivation relation* $L_1 \rightsquigarrow^\mu L_2$ over low level agents, is defined as the least relation satisfying the following axioms and inference rules.

(Act) $\mu F \overset{\mu}{\rightsquigarrow} F$, $\mu F \overset{\mu}{\rightsquigarrow} *F$

(Res) $L_1 \overset{\mu}{\rightsquigarrow} L_2$ implies $L_1\backslash\alpha \overset{\mu}{\rightsquigarrow} L_2\backslash\alpha$, $\mu \neq \alpha, \alpha^-$

(Rel) $L_1 \overset{\mu}{\rightsquigarrow} L_2$ implies $L_1[\phi] \overset{\phi(\mu)}{\rightsquigarrow} L_2[\phi]$

(Sum) $F_1 \overset{\mu}{\rightsquigarrow} L_2$ implies $F_1 + F \overset{\mu}{\rightsquigarrow} L_2$, $F + F_1 \overset{\mu}{\rightsquigarrow} L_2$,

(Com) $L_1 \overset{\mu}{\rightsquigarrow} L_2$ implies $L_1|F \overset{\mu}{\rightsquigarrow} L_2|F$, $F|L_1 \overset{\mu}{\rightsquigarrow} F|L_2$,

$F_1 \overset{\mu}{\rightsquigarrow} I_2$ implies $F_1|F \overset{\mu}{\rightsquigarrow} I_2|*F$, $F|F_1 \overset{\mu}{\rightsquigarrow} *F|I_2$.

$I_1 \overset{\mu}{\rightsquigarrow} I_2$ implies $I_1|I \overset{\mu}{\rightsquigarrow} I_2|I$, $I|I_1 \overset{\mu}{\rightsquigarrow} I|I_2$,

$F_1 \overset{\lambda}{\rightsquigarrow} L_2$ and $F_1' \overset{\bar{\lambda}}{\rightsquigarrow} L_2'$ implies $F_1|F_1' \overset{\tau}{\rightsquigarrow} L_2|L_2'$,

$I_1 \overset{\lambda}{\rightsquigarrow} L_2$ and $I_1' \overset{\bar{\lambda}}{\rightsquigarrow} L_2'$ implies $I_1|I_1' \overset{\tau}{\rightsquigarrow} L_2|L_2'$,

(Close) $L_1 \overset{\mu}{\rightsquigarrow} L_2$ implies $*L_1 \overset{\mu}{\rightsquigarrow} L_2$,

(Rec) $F[\mathrm{rec}\, x.F/x] \overset{\mu}{\rightsquigarrow} L$ implies $\mathrm{rec}\, x.F \overset{\mu}{\rightsquigarrow} L$.

Theorem 4.6. *Abstraction and composition commute for A^2CCS.*

Proof. Notice that higher (com) rules are exactly the (3.1) rules, and that lower (Com) rules correspond to the (3.3) rules with a type-driven definition of predicates P and Q. In fact, the first rule corresponds to entry $(- -/\mathbb{T}, x = \text{empty})$, the second to $(\mathbb{TF}/\mathbb{T}, x = *)$, the third to $(\mathbb{FF}/\mathbb{F}, x = \text{empty})$, the fourth to $(\mathbb{T}/\mathbb{T})$ and, finally, the fifth to $(\mathbb{F}/\mathbb{F})$. Function h_Q is given by the correspondence between visible low level agents and high level agents. Thus the existence of function h_T immediately follows from Theorem 3.2. □

The following theorem states that abstraction also commutes with all the other operators of the language.

Theorem 4.7 (High level is an abstraction of low level). *High level A^2CCS transition system is an abstraction of low level A^2CCS transition system.*

Proof. Function h_Q is given by the one-to-one correspondence between (actually, the identity of) visible low level agents and high level agents. To define h_T, given a metatransition m, we construct the corresponding transition $h_T(m)$. The correspondence is established by means of inference rules, whose correctness is obvious in all cases except for the synchronization rule. The proof is by induction on the length of the metatransition and on the depth of the proof tree of the first transition in the metatransition.

(act) $\mu F \overset{\mu}{\rightsquigarrow} F$ generates $\mu F \overset{\mu}{\rightsquigarrow} F$,

(seq) $F_1 \overset{\mu_1}{\rightsquigarrow} I_1 \overset{\mu_2}{\rightsquigarrow} I_2 \ldots I_{n-1} \overset{\mu_n}{\rightsquigarrow} F_2$ generates $F_1 \xrightarrow{\mu_1 \cdots \mu_n} F_2$
implies

$\mu F_1 \overset{\mu}{\rightsquigarrow} *F_1 \overset{\mu_1}{\rightsquigarrow} I_1 \ldots I_{n-1} \overset{\mu_n}{\rightsquigarrow} F_2$ generates $\mu F_1 \xrightarrow{\mu\mu_1 \cdots \mu_n} F_2$.

Transition $\mu F_1 \rightsquigarrow^{\mu} *F_1$ is proved by second axiom (Act), while the proof for $*F_1 \rightsquigarrow^{\mu_1} I_1$ is derived by that of $F_1 \rightsquigarrow^{\mu_1} I_1$ via (Close) rule. Note that the length of the metatransition in the premise of the rule is smaller. In rules (res), (rel), (sum) and (rec), the inductive step shortens the depth of the proof for the first transition in

the metatransition.

(res) $F_1 \overset{\mu_1}{\rightsquigarrow} I_1 \overset{\mu_2}{\rightsquigarrow} I_2 \dots I_{n-1} \overset{\mu_n}{\rightsquigarrow} F_2$ generates $F_1 \xrightarrow{\mu_1 \dots \mu_n} F_2$
and $\mu_i \notin \{\alpha, \alpha^-\}$, $i = 1, \dots, n$
implies

$F_1 \backslash \alpha \overset{\mu_1}{\rightsquigarrow} I_1 \backslash \alpha \dots I_{n-1} \backslash \alpha \overset{\mu_n}{\rightsquigarrow} F_2 \backslash \alpha$ generates $F_1 \backslash \alpha \xrightarrow{\mu_1 \dots \mu_n} F_2 \backslash \alpha$,

(rel) $F_1 \overset{\mu_1}{\rightsquigarrow} I_1 \overset{\mu_2}{\rightsquigarrow} I_2 \dots I_{n-1} \overset{\mu_n}{\rightsquigarrow} F_2$ generates $F_1 \xrightarrow{\mu_1 \dots \mu_n} F_2$
implies

$F_1[\phi] \overset{\phi(\mu_1)}{\rightsquigarrow} I_1[\phi] \dots I_{n-1}[\phi] \overset{\phi(\mu_n)}{\rightsquigarrow} F_2[\phi]$ generates $F_1[\phi] \xrightarrow{\phi(\mu_1 \dots \mu_n)} F_2[\phi]$,

(sum) $F_1 \overset{\mu_1}{\rightsquigarrow} I_1 \overset{\mu_2}{\rightsquigarrow} I_2 \dots I_{n-1} \overset{\mu_n}{\rightsquigarrow} F_2$ generates $F_1 \xrightarrow{\mu_1 \dots \mu_n} F_2$
implies

$F_1 + F \overset{\mu_1}{\rightsquigarrow} I_1 \overset{\mu_2}{\rightsquigarrow} I_2 \dots I_{n-1} \overset{\mu_n}{\rightsquigarrow} F_2$ generates $F_1 + F \xrightarrow{\mu_1 \dots \mu_n} F_2$
and

$F + F_1 \overset{\mu_1}{\rightsquigarrow} I_1 \overset{\mu_2}{\rightsquigarrow} I_2 \dots I_{n-1} \overset{\mu_n}{\rightsquigarrow} F_2$ generates $F + F_1 \xrightarrow{\mu_1 \dots \mu_n} F_2$,

(rec) $F_1[\mathrm{rec}\, x.F_1/x] \overset{\mu_1}{\rightsquigarrow} I_1 \dots I_{n-1} \overset{\mu_n}{\rightsquigarrow} F_2$ generates $F_1[\mathrm{rec}\, x.F_1/x] \xrightarrow{\mu_1 \dots \mu_n} F_2$
implies

$\mathrm{rec}\, x.F_1 \overset{\mu_1}{\rightsquigarrow} I_1 \dots I_{n-1} \overset{\mu_n}{\rightsquigarrow} F_2$ generates $\mathrm{rec}\, x.F_1 \xrightarrow{\mu_1 \dots \mu_n} F_2$,

(com) $F_1 \overset{\mu_1}{\rightsquigarrow} I_1 \overset{\mu_2}{\rightsquigarrow} I_2 \dots I_{n-1} \overset{\mu_n}{\rightsquigarrow} F_2$ generates $F_1 \xrightarrow{\mu_1 \dots \mu_n} F_2$
implies

$F_1 | F \overset{\mu_1}{\rightsquigarrow} I_1 | F \dots I_{n-1} | F \overset{\mu_n}{\rightsquigarrow} F_2 | F$ generates $F_1 | F \xrightarrow{\mu_1 \dots \mu_n} F_2 | F$
and

$F | F_1 \overset{\mu_1}{\rightsquigarrow} F | I_1 \dots F | I_{n-1} \overset{\mu_n}{\rightsquigarrow} F | F_2$ generates $F | F_1 \xrightarrow{\mu_1 \dots \mu_n} F | F_2$,

$F_1' \overset{\mu_1'}{\rightsquigarrow} I_1' \overset{\mu_2'}{\rightsquigarrow} I_2' \dots I_{j-1}' \overset{\mu_j'}{\rightsquigarrow} F_1'$ generates $F_1' \xrightarrow{\mu_1' \dots \mu_j'} F_2'$
and

$F_1'' \overset{\mu_1''}{\rightsquigarrow} I_1'' \overset{\mu_2''}{\rightsquigarrow} I_2'' \dots I_{k-1}'' \overset{\mu_k''}{\rightsquigarrow} F_2''$ generates $F_1'' \xrightarrow{\mu_1'' \dots \mu_k''} F_2''$
and

S-syn$(\mu_1' \dots \mu_j', \mu_1'' \dots \mu_k'', \mu_1 \dots \mu_n)$
implies

$F_1' | F_1'' \overset{\mu_1}{\rightsquigarrow} I_1 \overset{\mu_2}{\rightsquigarrow} I_2 \dots I_{n-1} \overset{\mu_n}{\rightsquigarrow} F_2' | F_2''$ generates $F_1' | F_1'' \xrightarrow{\mu_1 \dots \mu_n} F_2' | F_2''$.

Intermediate states I_i are not described with respect to their components because this information depends on the proof for S-syn. Thus the correctness of the synchronization rule is proved by resorting to Theorem 4.6.

To prove that function h_T is surjective, given a high level transition, we have to construct a corresponding metatransition. We simply notice that the previous inference rules can be restated by swapping the two arguments of the predicate 'generates'.

For instance,

(act) $\mu F \xrightarrow{\mu} F$ generates $\mu F \overset{\mu}{\rightsquigarrow} F$,

(seq) $F_1 \xrightarrow{\mu_1 \cdots \mu_n} F_2$ generates $F_1 \overset{\mu_1}{\rightsquigarrow} I_1 \overset{\mu_2}{\rightsquigarrow} I_2 \ldots I_{n-1} \overset{\mu_n}{\rightsquigarrow} F_2$

implies

$\underline{\mu} F_1 \xrightarrow{\mu\mu_1 \cdots \mu_n} F_2$ generates $\underline{\mu} F_1 \overset{\mu}{\rightsquigarrow} *F_1 \overset{\mu_1}{\rightsquigarrow} I_1 \ldots I_{n-1} \overset{\mu_n}{\rightsquigarrow} F_2$,

and so on. □

Theorem 4.8 (Semantic extension of CCS). *Both high level A^2CCS derivation relation $F_1 \rightarrow^s F_2$ and low level derivation relation $L_1 \rightsquigarrow^{\mu} L_2$, when restricted to CCS agents, are Milner's derivation relation.*

Proof. When no private prefixes are present, the upper rule seq is never used, and thus the sequences of actions are of unitary length ($s = \mu$). Similarly, no lower transition exists between a visible and an invisible state. □

5. An example

We solve the deadlock problem for the dining philosophers by programming the acquisition of the forks as an atomic behaviour. We restrict our attention to two philosophers and two forks only. A philosopher T and a fork D are described by means of A^2CCS terms as follows:

$$T = \text{rec } x.\ \underline{\text{up}}\ \text{up down down } x;$$

$$D = \text{rec } x.\ \text{up}^-\ \text{down}^-\ x.$$

The corresponding low level transition systems are shown in Fig. 2. The whole system is Dinner $= ((T|D)|(T|D))\backslash\text{up}\backslash\text{down}$. In Figs. 3 and 4 we see the low level and the high level transition system for Dinner.

The correspondence between lower and higher systems can be better understood by means of the following example. Let us consider the metatransition

$$((T|D)|(T|D)) \overset{\tau}{\rightsquigarrow} ((*\text{up down down } T|*D)|(T|\text{down}^-\ D))$$

$$\overset{\tau}{\rightsquigarrow} ((\text{down down } T|\text{down}^-\ D)|(T|\text{down}^-\ D))$$

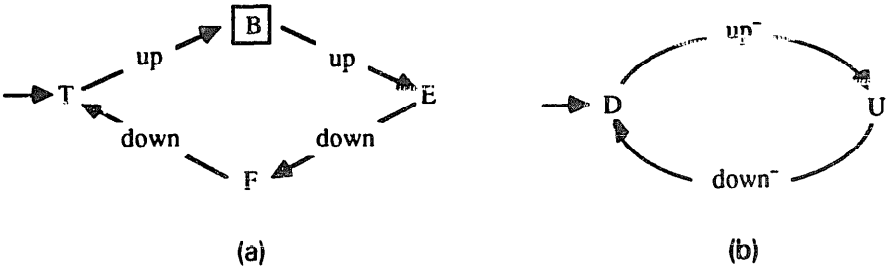

Fig. 2. Philosopher (a) and fork (b) low level transition systems. The state in a box is invisible. Philosopher condition of thinking, beginning the dinner (one fork up), eating and finishing the dinner (one fork down) are associated to the states T, B, E and F, respectively. Similarly, forks may be in two positions: down (D) and up (U).

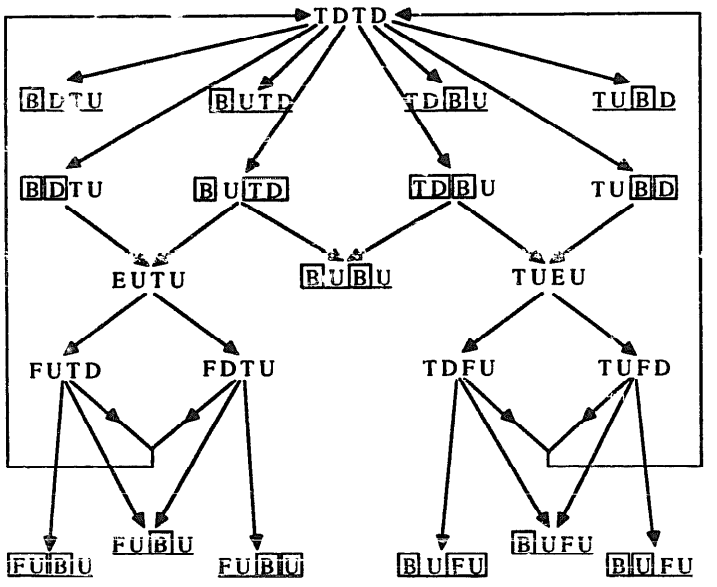

Fig. 3. The low level transition system for Dinner. States with boxes are invisible and each box represents the range of a mark; for instance $\boxed{B}U\boxed{TD}$ represents

$$((*(\text{up down down } T)|(\text{down}^- D))|*(T|D))\backslash\text{up}\backslash\text{down}.$$

Underlined states correspond to deadlock states. All transitions are labelled by τ.

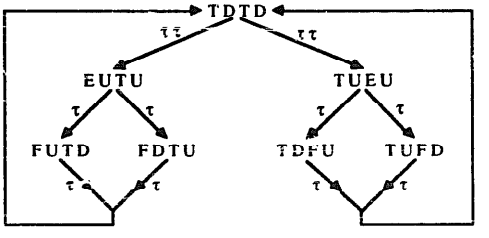

Fig. 4. The high level system for Dinner. The two transactions are labelled by $\tau\tau$.

where all the states are restricted with respect to actions up and down. The corresponding transition in the high level transition system is

$$((T|D)|(T|D)) \xrightarrow{\tau\tau} ((\text{down down } T|\text{down}^- D)|(T|\text{down}^- D)),$$

which is also the image of the metatransition

$$((T|D)|(T|D)) \overset{\tau}{\rightsquigarrow} ((*\text{up down down } T|\text{down}^- D)|*(T|D))$$

$$\overset{\tau}{\rightsquigarrow} ((\text{down down } T|\text{down}^- D)|(T|\text{down}^- D)).$$

These two metatransitions correspond to the same higher transition, which does not differentiate the acquisition order of the forks. Moreover, a sequence of lower

transitions which is not a metatransition has not a corresponding higher transition. For instance, the sequence

$$((T|D)|(T|D))$$
$$\overset{\tau}{\rightsquigarrow} ((*\text{up down down } T|\text{down}^- D)|*(T|D))$$
$$\overset{\tau}{\rightsquigarrow} ((*\text{up down down } T|\text{down}^- D)|(*\text{up down down } T|\text{down}^- D))$$

represents the situation (deadlock!) in which both philosophers have taken one fork. This sequence is not a metatransition and thus no higher transition will correspond to it.

Note that all deadlock states are invisible. Note also that, due to the definition of parallel composition, it is possible to reach states which are very similar from a syntactical point of view, but with really a different behaviour. For instance, the two states

$$((*\text{up down down } T|*D)|(T|\text{down}^- D))$$

and

$$((*\text{up down down } T|D)|(T|\text{down}^- D))$$

which can be both reached from state $(T|D)|(T|D)$ thanks to a proof involving the second and the first (Com) rule, respectively, behave quite differently, the latter being a completely deadlocked state.

6. Related works

The problem of defining an abstraction mechanism in concurrent, possibly distributed systems has been considered by many researchers. There are essentially two different methodologies to specify a system: describing an abstract model of system behaviour (the so-called *constructive* approach), or stating the properties the system should have (the *axiomatic* approach). The bulk of papers on the problem of hierarchical specification is on the lines of the axiomatic approach (e.g. [1, 8, 9]), while the present paper, based on CCS, is along the lines of the constructive approach, together with [2, 4, 5, 13]. On the other hand, there is another obvious way to partition these models: those based on a shared memory model of computation [1, 3, 8–11] and those based on a message passing model of computation [2, 12, 13]. Papers [4, 5] are based on the constructive approach, and provide a suggestive bridge between shared memory and message passing models of computation. Our paper, like [2, 13], follows the constructive approach using a message passing model. However, neither [2] nor [13] considers, as we do, two distinct

models at different level of abstraction. Furthermore, the language considered in [2] is much more restrictive than ours, and the semantics defined in [13] is in our view sometimes inadequate. We do not know of any paper following the axiomatic approach and using a message passing model of computation.

A particular, but very important problem related to atomic actions is the problem of serializability of (nested) transactions in data base theory [3]. However, the notion of transaction in data base theory is broader than ours in that it only requires *virtual* atomicity: in an actual, efficient implementation, the scheduler is allowed to interleave distinct transactions, but it operates guaranteeing the same results as if the transactions were not interleaved, namely as if the scheduler were serial. Instead, we only allow interleaving of subtransactions of the same transaction, and the notion of communication in the model is based on synchronous message passing. For a model of nested transactions based on I/O automata, a(nother!) simple formalization of communicating automata, see [10]. Starting from that model, in [11] a notion of abstraction is defined in terms of the notion of *possibilities mapping* between automata, relating the states of one automaton to the states of the other; finally, a notion reminiscent of bisimulation is introduced to verify that the two automata exhibit the same behaviour. This work is in a sense constructive, but without the simplicity of Plotkin's Structural Operational Semantics technique [14].

Finally, we want to compare our language with a similar one [13], strongly based on [2]. There, an atomic calculus of communicating systems (ATCCS) is defined, introducing a new operator, represented by $*$, whose intuitive meaning is *strong sequencing*. For instance, $a*b\mathrm{Nil}$ means that the execution of action b must immediately follow the execution of action a. So we may expect that $a*b\mathrm{Nil}$ represents an atomic action. Moreover, the arrows are enriched with an attribute which has two possible values: "at" to indicate an atomic derivation and "int" to indicate an interruptible derivation. The relevant rules for that calculus are the following:

Sequencing:

$$a*A \xrightarrow{a\text{-(at)}} A$$

$$aA \xrightarrow{a\text{-(int)}} A$$

Composition:

$$A \xrightarrow{a\text{-(at)}} A' \quad \text{implies} \quad A|B \xrightarrow{a\text{-(at)}} A'|_L B$$

$$B \xrightarrow{a\text{-(at)}} B' \quad \text{implies} \quad A|B \xrightarrow{a\text{-(at)}} A|_R B'$$

$$A \xrightarrow{a\text{-(int)}} A' \quad \text{implies} \quad A|B \xrightarrow{a\text{-(int)}} A'|B$$

$$B \xrightarrow{a\text{-(int)}} B' \quad \text{implies} \quad A|B \xrightarrow{a\text{-(int)}} A|B'$$

$$A \xrightarrow{a\text{-(any)}} A' \text{ and } B \xrightarrow{a\text{-(any)}} B' \quad \text{implies} \quad A|B \xrightarrow{i\text{-(any)}} A'|B'$$

where $|_L$ and $|_R$ are the auxiliary composition operators, with the requirement that

the next action has to be taken, respectively, from the left or the right component, and where "any" is a variable that ranges over {at, int}.

Left and right composition:

$$A \xrightarrow{a\text{-(at)}} A' \text{ implies } A|_L B \xrightarrow{a\text{-(at)}} A'|_L B$$

$$B \xrightarrow{a\text{-(at)}} B' \text{ implies } A|_R B \xrightarrow{a\text{-(at)}} A|_R B'$$

$$A \xrightarrow{a\text{-(int)}} A' \text{ implies } A|_L B \xrightarrow{a\text{-(int)}} A'|B$$

$$B \xrightarrow{a\text{-(int)}} B' \text{ implies } A|_R B \xrightarrow{a\text{-(int)}} A|B'$$

Even if the calculus has some nice properties (e.g. associativity of composition, see the section on conclusions for a discussion about this topic), apparently there are some deep inadequacies. First of all, this is not really the correct notion of atomic action, in the sense that there is no way to prescribe that a process has to execute a sequence of actions in an atomic manner, whatever the context is. In fact, the semantic definition allows manners of composing systems which are unreasonable, if the main goal is defining a system at two different levels of abstraction. In particular, the basic underlying requirement that the composition of atomic actions has to be atomic, is disregarded. As a consequence, there is no way to specify a high level transition system for ATCCS. To be more precise, consider the following example. Consider the ATCCS terms $A = a*b\text{Nil}$, $B = a*d\,\text{Nil}$ and $A' = c\text{Nil}$. Their execution in isolation gives rise to atomic actions ab, ad and c, respectively. However, term $A|B$ can execute a sequence of actions idb which is not atomic: the composition of ab and ad is not an atomic behaviour. In fact, if we consider term $(A|B)|A'$, we see that it can execute the nonatomic sequence $idcb$, i.e. idb can be interrupted and intermediate states are observable:

$$(a*b\text{Nil}|a*d\,\text{Nil})|c\text{Nil} \xrightarrow{i\text{-(at)}} (b\text{Nil}|d\,\text{Nil})|_L c\text{Nil} \xrightarrow{d\text{-(int)}} (b\text{Nil}|\text{Nil})|c\text{Nil}$$

$$(b\text{Nil}|\text{Nil})|c\text{Nil} \xrightarrow{c\text{-(int)}} (b\text{Nil}|\text{Nil})|\text{Nil} \xrightarrow{b\text{-(int)}} (\text{Nil}|\text{Nil})|\text{Nil}.$$

Notice that this counterintuitive behaviour is allowed since an agent (in this example $A|B$) may execute an interruptible transition (here labelled by d) even if one of its subagents (A) is still executing an atomic action (ab). In A^{2}CCS, instead, all the subagents of an agent that have just executed a commit transition, are visible.

A further inadequacy is that a synchronization between an interruptible and an atomic transition is apparently forbidden. Thus, for instance, $a\text{Nil}|a*b\text{Nil}$ cannot synchronize.

ACMP [2] is restrictive in that it requires synchronous cooperation, namely two agents involved in the same "tight region" cannot interleave. Furthermore, the sequence of two or more tight regions can be composed with a single tight region,

so that the constuction of a higher level transition system becomes problematic: one should be able to compose not only two transitions but also two computations even in the higher transition system. For instance, agent $\alpha:\alpha\mathrm{Nil}|\alpha^{-}\alpha^{-}\mathrm{Nil}$, where : is the operator of strong sequencing, may perform atomic action $\tau\tau$, composing a single atomic action (left component) with a two step computation (right component).

7. Conclusion

We have modelled atomic behaviours of concurrent systems and their implementations using high level and low level transition systems. A syntactic and semantic extension of CCS has also been proposed, which might be useful as a specification language for synchronization problems in data and resource management.

Finally, some methodological comments. The technical problem which underlies the present research is the definition of a suitable notion of a low level transition system. Extra states are needed for representing the intermediate states of an atomic behaviour, and we resort to marks to represent them. A predicate checking state visibility is introduced. It is obviously defined by structural induction, and is used in (3.3) to define the operator of parallel composition, whose introduction thus requires an extra subproof. However, the notion of visibility allows us to define easily the mapping between lower and higher transition systems. An alternative solution, proposed in [2], could consist of introducing new operators such as *left*, *right* and *communication merge*, thus also adding extra states. However, it is not clear how to associate a notion of visibility to the states obtained in this way. Thus, it is not obvious how their methodology could be extended to cope with the issue of hierarchical specification.

Some problems are still open. For instance, if in the example of the previous section we compose philosophers and forks in a different way

$$\mathrm{Dinner}' = ((T|T)|(D|D))\backslash\mathrm{up}\backslash\mathrm{down},$$

we obtain a one-state, completely deadlocked high level transition system! An explanation may be as follows. To work correctly, subsystem $T|T$ would need that its partner could perform a transition labelled $\mathrm{up}^{-}\mathrm{up}^{-}$. On the contrary, subsystem $D|D$ is not able to do so, since it can execute instead a sequence of two transitions, each labelled up^{-}.

The loss of the associativity property for the composition operator may be considered intrinsic to the hierarchical construction needed for handling nested atomic transitions. A possible alternative could be to have two distinct composition operators: an associative operator of parallel composition with the usual semantics, and another operator, which may be called *block*, which works as an atomic transition manager.

References

[1] H. Barringer, R. Kuiper and A. Pnueli, Now you may compose temporal logic specifications, in: *Proc. 16th ACM Symp. on Theory of Computation* (1984) 51-63.

[2] J.A. Bergstra and J.W. Klop, Process algebra for synchronous communication, *Inform. and Control* **60** (1984) 109-137.

[3] P.A. Bernstein and N. Goodman, Concurrency control in distributed database systems, *ACM Comput. Surv.* **13** (1981) 185-222.

[4] G. Boudol, Communication is an abstraction, INRIA Rapport de Recherche No. 636, March 1987.

[5] G. Boudol and I. Castellani, Concurrency and atomicity, *Theoret. Comput. Sci.* **59** (1988) 25-84.

[6] L. Castellano, G. De Michelis and L. Pomello, Concurrency vs. interleaving: an instructive example, *Bull. Eur. Assoc. Theoret. Comput. Sci.* **31** (1987) 12-15.

[7] W.P. De Roever, Questions to Robin Milner—A responder's commentary, in: *Proc. IFIP Congress '86* (1986) 515-518.

[8] L. Lamport, Specifying concurrent program modules, *ACM Trans. Programming Languages Systems* **5** (1983) 190-222.

[9] L. Lamport, What good is temporal logic?, in: *Proc. IFIP Congress '83* (1983) 657-668.

[10] N. Lynch and M. Merrit, Introduction to the theory of nested transactions, in: *Proc. Internat. Conf. on Data Base Theory '86*, Lecture Notes in Computer Science **243** (Springer, Berlin, 1986) 278-305.

[11] N. Lynch and M. Tuttle, Hierarchical correctness proofs for distributed algorithms, in: *Proc. 6th ACM Symp. on Principles of Distributed Computing*, Vancouver (1987).

[12] R. Milner, Lectures on a calculus for communicating systems, in: M. Broy, ed., *Control Flow and Data Flow: Concepts of Distributed Programming*, NATO ASI Series F14 (1985).

[13] A. Obaid and L. Logrippo, An atomic calculus of communicating systems, in: *IFIP WG 6.1, 7th Conf. on Protocol Specification, Testing and Verification*, Zurich (1987).

[14] G. Plotkin, A structural approach to operational semantics, Technical Report DAIMI FN-19, Aarhus University, Department of Computer Science, Aarhus, 1981.